SOFTWARE MAINTENANCE: RESEARCH AND PRACTICE, VOL. 9, 161–175 (1997)

Research

A Method for Estimating Maintenance Cost in a Software Project: A Case Study

JUAN CARLOS GRANJA-ALVAREZ[1]* AND MANUEL JOSÉ BARRANCO-GARCÍA[2]

[1]*Department of Computer Science, University of Granada, Avda. de Andalucía 38, E18071 Granada, Spain*
[2]*Microjisa Inc., Calle Pintor Manuel Angeles Ortiz, 2, Bajo, E23006 Jaén, Spain*

SUMMARY

Various research works indicate that the maintenance stage consumes most of the resources needed for a software project. Thus, this stage must be especially considered in productivity studies. Maintainability is the quality factor including all those software characteristics designed to make the product easier to maintain towards the end of achieving greater productivity in the maintenance stage. This paper proposes an empirical model for estimating maintenance cost based on this quality factor, as well as the method of using it. Finally, a practical case will be considered which reinforces the validity of this model. © **1997 by John Wiley & Sons, Ltd.**

J. Softw. Maint., **9**, 161–175 (1997)

No. of Figures: 1. No. of Tables: 3. No. of References: 22.

KEY WORDS: maintainability; cost estimation; productivity; software metrics; case study; software maintenance

1. INTRODUCTION

1.1. Background

Frequently, software companies look for maximum productivity while developing their products, but leave the maintenance stage in second place. This is an error, because, as much experience has revealed, this is the very stage which consumes the greater portion of resources, more than 60% (Canning, 1972; Harrison and Cook, 1990; Nosek and Palvia, 1990).

Apart from the obvious economic implications, if productivity in the maintenance stage is low, the persons employed to develop the project may need to invest much of their time in later maintenance. Such is the experience which the authors have gleaned from previous projects and from the application of their estimating methods to real projects.

* Correspondence to: Juan Carlos Granja-Alvarez, Department of Computer Science, University of Granada, Avda de Andulucía 38, E18071-Granada, Spain. E-mail: jcgranja@goliat.urg.es

CCC 1040–550X/97/030161–15$17.50

Received 28 February 1996
Revised 1 December 1996

Consequently, if a company wishes to take on further projects, it must include an entirely new team among its staff. This means at least a partial loss from the experience gathered by the first team, becoming unavailable for further projects. Furthermore, the new team would need to be trained in methods and tools used by the software company. This has implications for validation efforts (Schneidewind, 1987).

1.2. Definition of software maintenance

Software maintenance is defined as the process of modifying, for update or repair, existing operational software, but leaving its primary functions intact (Boehm, 1981, pp. 54–55). This definition excludes major enhancements (Boehm, 1981, pp. 534–535) and hence differs from Swanson's typology (Swanson 1976; Swanson and Chapin, 1995). Over the years, several software maintenance models have been proposed, to emphasize particular aspects of software maintenance. Boehm's maintenance model consists of three major phases: understanding, modifying and validating the software. Lewis and Henry (1989) have addressed the need for metrics during development to help make the resulting product more maintainable. Productivity in the maintenance stage is directly related to the maintainability of the product. Current maintenance distribution models can be categorized into two general schools of thought. One regards software maintenance as bearing unique features (Chapin, 1987, 1988; Layzell and Loucopopulos, 1988), while a second treats maintenance as essentially the same as testing (Musa, Iannino and Okumoto, 1990; Skramstad and Khan, 1992).

1.3. Definition of software maintainability

Maintainability is the quality factor which includes all the features of the software to make it easier to maintain or which make the maintenance stage more productive (Frost, 1985). The question here is to ascertain the efficiency of the maintenance process by using the body of data collected in previous projects. These objective and validated data, when applied via the proposed techniques, conserve the maintainability characteristics while the product is being developed by providing for the needs forecast for the maintenance stage, and so avoid unnecessary future costs.

This paper proposes a model for estimating maintenance cost, based on the experience gained from previous projects. We take the COCOMO (Boehm, 1981) as the basis of our model, in which we incorporate indices measuring the maintainability of the product. The results of this research have been tested in practice; we summarize a report near the end of this paper. The practical application with empirical data has been carried out and reported elsewhere (Barranco-García and Granja-Alvarez, 1996).

2. COCOMO IN THE MAINTENANCE STAGE

COCOMO (*Co*nstructive *Co*st *Mo*del) is a model for estimating costs of software projects. It was created by Barry W. Boehm and published in 1981 using data collected from 63 projects. The quantity of projects studied and the care in its articulation have made the model popular and encouraged its use. The model offers empirical formulaes for estimating

software costs. We have taken the COCOMO as the starting point for our research, although we have borne in mind other data from the same author (Boehm, 1976, 1987).

After application of the first version of his model to a wide range of situations, Boehm concluded that coverage for only one mode of developing software was insufficient, so he provided for three modes: organic, semidetached and embedded. In this way, he recognized the influence of various characteristics such as size, communication needs, experience in similar projects, etc.

Furthermore, Boehm provides the COCOMO in three versions: basic, intermediate, and detailed. The basic version is useful for quick estimations, although without fine precision. The intermediate version deals with 15 attributes of the project (reliability required, database size, memory restrictions, response time required, etc.), all of whose valuation acts as a multiplying factor in the model. The detailed version deals with estimates in each of the phases in the life cycle of the project.

The basic version of the model offers the following formulaes to calculate the cost of software development measured in *MM* (i.e., man-months):

$$\text{organic mode} \qquad MM_{\text{DEV}} = 2{\cdot}4\ KS^{1{\cdot}05} \tag{1}$$

$$\text{semidetached mode} \qquad MM_{\text{DEV}} = 3{\cdot}0\ KS^{1{\cdot}12} \tag{2}$$

$$\text{embedded mode} \qquad MM_{\text{DEV}} = 3{\cdot}6\ KS^{1{\cdot}20} \tag{3}$$

where *KS* is an estimate of the delivered program size in thousands of instructions (or roughly, lines of source code).

To estimate the maintenance cost, another parameter is needed: the annual change traffic (*ACT*) which consists of the proportion of original instructions that undergo a change during a year by addition or modification.

$$ACT = \frac{NNL + NML}{NOL} \tag{4}$$

where:

NNL = number of new lines,

NML = number of modified lines, and

NOL = number of original lines.

In the COCOMO, the cost in the maintenance stage is given by the product of the development cost and the annual change traffic:

$$MM_{\text{MAIN}} = ACT \bullet MM_{\text{DEV}} \tag{5}$$

3. MODEL FOR THE ESTIMATION OF MAINTENANCE COST

3.1. Maintainability index

Maintainability is, beyond doubt, the software quality factor with the most influence in the maintenance stage. A study made by W. Itzfeld in Germany and compiled by Wallmüller (1994) presents quality metrics ranking in which maintainability metrics were reported in first position by 67% of those asked. Using the definition of maintenance we summarized earlier, Boehm (1979) recognized the importance of maintainability. One of his studies indicated that maintenance costs for software with low maintainability had a relation of 40 to 1 with respect to new development. It is obvious that an interdependent relationship exists between maintainability characteristics of developed software and maintenance cost.

So, in order to calculate the estimated maintenance cost we must consider a factor that indicates a measurement of the maintainability (facile in respect to maintenance) of the product. Taking as a starting point the formula in expression (5) for estimating maintenance cost from the COCOMO, we will include the factor we call the 'maintainability index', which will be a function of some empirical software measurements, as follows:

$$MM_{\mathrm{MAIN}} = ACT \bullet MM_{\mathrm{DEV}} \bullet I_{\mathrm{MAIN}} \tag{6}$$

$$I_{\mathrm{MAIN}} = f\,(X_1, X_2, \ldots, X_n) \tag{7}$$

This I_{MAIN} index will show the inverse of the degree of maintainability of the product, where high values of this index indicate low maintainability and low values indicate high maintainability. Also it will be a good indication of the productivity in the maintenance stage. This maintainability index is not the same as the maintainability index described by others (Welker, Oman and Atkinson, 1997; West, 1993; Peercy, 1996).

Consequently our main aim must be to determine what our maintainability index will be. That is to say, we must ascertain the relationship between the estimated maintenance cost and those characteristics which make a program more or less maintainable.

There are two steps to formulate this model:

(1) Establish maintainability measurements.
(2) Obtain the maintainability functions which relate the established metrics and the maintainability index.

Before taking up these two points we should bear in mind the three main activities which occur in maintenance. To reflect them, the maintainability index is divided into three component indices: an understandability index, a modifiability index and a testability index.

3.2. Maintainability components

As just suggested, maintenance action can be divide into three parts:

- *understanding* the changes to be made,

- *modifying* or making the change, and
- *testing,* or verifying the changes made.

These are clearly differentiated and performed one after the other, so the maintenance cost could be considered to be the sum of three costs expressed in man-months: understanding, modifying and testing:

$$MM_{\mathrm{MANT}} = MM_{\mathrm{U}} + MM_{\mathrm{M}} + MM_{\mathrm{T}} \tag{8}$$

As a result from expressions (6) and (8), we will have three maintainability indexes, I_{U}, I_{M} and I_{T}, which relate the two parameters of a software project, ACT and MM_{DEV}, to the three components of maintenance cost expressed in man-months:

$$MM_{\mathrm{U}} = ACT \bullet MM_{\mathrm{DEV}} \bullet I_{\mathrm{U}} \tag{9}$$

$$MM_{\mathrm{M}} = ACT \bullet MM_{\mathrm{DEV}} \bullet I_{\mathrm{M}} \tag{10}$$

$$MM_{\mathrm{T}} = ACT \bullet MM_{\mathrm{DEV}} \bullet I_{\mathrm{T}} \tag{11}$$

Consequently and in a parallel manner from expressions (7) and (8), the maintainability index, I_{MAIN}, will be given as the sum of these three equally weighted indices which may have very differing values,

$$I_{\mathrm{MAIN}} = I_{\mathrm{U}} + I_{\mathrm{M}} + I_{\mathrm{T}} \tag{12}$$

where:

I_{MAIN} = maintainability index,
I_{U} = understandability index,
I_{M} = modifiability index, and
I_{T} = testability index.

Hence from expressions (6), (8) and (12), the total maintenance cost is:

$$MM_{\mathrm{MAIN}} = MM_{\mathrm{U}} + MM_{\mathrm{M}} + MM_{\mathrm{T}} = ACT \bullet MM_{\mathrm{DEV}} \bullet (I_{\mathrm{U}} + I_{\mathrm{M}} + I_{\mathrm{T}}) \tag{13}$$

3.3. Maintainability metrices

The model proposed here, which has been used in case studies, considers only three software characteristics. Each one directly affects one maintainability component.

X_{U}: understandability metric

The number of comment lines for every 100 lines of code. We observe that there is a close relationship between the internal documentation of the code and understanding cost.

As expected, as the value of the understandability metric increases (number of comment lines), the understandability index (directly proportional to understandability cost) decreases.

X_M: *modifiability metric*

The number of lines without constant data for every 100 lines of code. We observe that the more numerous the constant data in the code, the bigger the modification cost.

X_T: *testability metric*

The number of error testing lines for every 100 lines of code. We observe that testing the code will be simpler if there are error detection and treatment procedures built into the code.

These three characteristics have been chosen because we observe that (1) they are easily measured, and (2) they have a great influence on maintainability. Nevertheless, the model can be applied whatever the metrics chosen, provided the interdependence between each metric and its maintainability component can be demonstrated.

3.4. Maintainability functions

Elaborating expression (12) to incorporate the maintainability metrics, we introduce the maintainability function F. As described later in Section 3.5.2, this is a statistically determined relationship between the metrics X_U, X_M and X_T just described, and the indices I_U, I_M and I_T, and can be summarized as follows:

$$I_U = F_U (X_U) \tag{14}$$

$$I_M = F_M (X_M) \tag{15}$$

$$I_T = F_T (X_T) \tag{16}$$

To obtain these F functions, it is necessary to use something fundamental to all estimation processes, historical data. The experience acquired in former projects is of great value in estimating new projects. Thus, the management procedures of the software project must include mechanisms which allow us to take these measurements:

(a) of the developed product:
  X_U: understandability metric,
  X_M: modifiability metric, and
  X_T: testability metric;
(b) of the maintenance process:
  MM_U: understanding cost expressed in man-months,
  MM_M: modifying cost expressed in man-months, and
  MM_T: testing cost expressed in man-months.

The measurements of the maintenance process must be made annually. Every year, the annual change traffic (*ACT*) experienced must be determined, and with that, the cost expended in each task (understanding MM_U, modifying MM_M and testing MM_T) must be measured in man-months for all of the *ACT*. Maintainability indexes can then be obtained from the measured maintenance efforts using a simple formula. For example and consistent with expression (13), for the understandability index, the formula that implements expression (14) using historical data is:

$$I_U = \frac{MM_U}{ACT \bullet MM_{DEV}} \tag{17}$$

where:

I_U = understandability index,

MM_U = maintenance understanding cost expressed in man-months,

ACT = annual change traffic, and

MM_{DEV} = development cost expressed in man-months.

In a parallel manner, we obtain numeric values for I_M and I_T by using the modifying and testing efforts respectively, project by project.

All the data necessary to apply the model just described are collected and updated annually in what we call a history table (HT) with the structure of Table 1.
The abbreviations used are as follows:

P_i	Identification of the project.
ACT_i	Average annual value of *ACT* for project *i*, using expression (4).
MM_{DEV}	Cost of development of the product (measured in man-months)
X_U	Value of the understandability metric for the project's product.
MM_U	Average annual cost of understanding (measured in man-months).
I_U	Understandability index, using expression (17).
X_M	Value of the modifiability metric for the project's product.
MM_M	Average annual cost of modification (measured in man-months).
I_M	Modifiability index, determined parallel to expression (17).
X_T	Value of the testability metric for the project's product.

Table 1. Form and content of the history table

Project	*ACT*	MM_{DEV}	X_U	MM_U	I_U	X_M	MM_M	I_M	X_T	MM_T	I_T
P_1	ACT_1	MM_{DEV1}	X_{U1}	MM_{U1}	I_{U1}	X_{M1}	MM_{M1}	I_{M1}	X_{T1}	MM_{T1}	I_{T1}
…	…	…	…	…	…	…	…	…	…	…	…
P_n	ACT_n	MM_{DEVn}	X_{Un}	MM_{Un}	I_{Un}	X_{Mn}	MM_{Mn}	I_{Mn}	X_{Tn}	MM_{Tn}	I_{Tn}

MM_T Average annual cost of testing (measured in man-months).
I_T Testability index, determined parallel to expression (17).

3.5. Implementation method

3.5.1. Overview

In this section a method of introducing the model in a software project is described. Figure 1 shows the elements and processes which have a role in the model. As we can see from this figure, there are three processes which use the historical data from the HT.

3.5.2. Model adjustment

Model adjustment deals with determining what the maintainability functions will be (see Section 3.4). Taking from the HT a two-column subtable formed from columns X_U and I_U we have a set of points which can be represented two-dimensionally $\{(X_{Ui}, I_{Ui})/i = 1 \ldots n\}$. Making a simple regression analysis on this group of points, we can obtain the best fit curve. The function F_U thus obtained, which we call the understandability function, will allow us to estimate the understandability index for a given value of the X_U metric, following expression (14). In the same way, we arrive at the modifiability function, F_M, and the testability function, F_T, from the set of points $\{(X_{Mi}, I_{Mi})/i = 1 \ldots n\}$ and $(X_{Ti}, I_{Ti})/i = 1 \ldots n\}$ respectively, using expressions (15) and (16).

The adjustment method used here is the method of least squares. For example, the

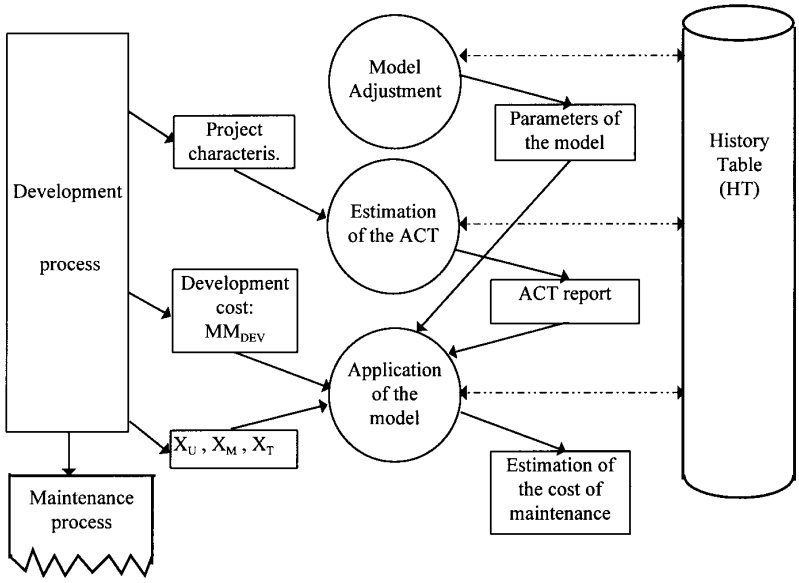

Figure 1. Plan of the method of model implementation

understandability function, F_U, is such that the sum of the squares of the errors is a minimum. That is to say, given expression (14):

$$\sum_{i=1}^{n} e_{Ui}^2 \text{ is minimum, where } e_{Ui} = |I_{Ui} - F_U(X_{Ui})| \tag{18}$$

Consequently there are two obvious models for analysing the regression of the set of points $\{(X_U, I_U)\}$: the linear model of negative slope, and the negative exponential model. Of the two models, the negative exponential is the more suitable because, normally, the improvement in understanding that a new comment line gives, will be greater the smaller the concentration of comments lines in the program. This is well understood if we take it to the limits—that is, seeing what happens when a comment line is added to a program with no internal documentation, or to a program with complete internal documentation. It is obvious that understanding in the first case will be greatly improved over that in the second case.

3.5.3. Estimation of the future ACT

The annual change traffic (*ACT*) described in expression (4) is a look backward in time. For our model, we need a look forward in time, an estimate of the future *ACT* for any project of concern. Making such an estimate is a process which needs to be based on experience. There are two basic elements in the whole estimation process: historical data and expert judgement. The method proposed here uses two elements to make an estimation.

We begin with the existence of a series of given characteristics of a software project (not to be mistaken for product characteristics). The characteristics must be believed to be determinants of or important influences upon the annual change traffic (*ACT*). The characteristics should be evaluated and revised periodically by the company's experts to identify critical characteristics of the projects covered in the HT (history table). Based upon the data in the HT, each characteristic will be assigned a weight p_j which permits us to give appropriate recognition to every characteristic based upon accumulated *ACT* data. Each project will only have two possibilities for every characteristic—that is, to have it or not. So, for a table formed by n projects, we will have for each project the data shown by four matrices. Matrices **A** and **T** consist of historical data; matrices **C** and **B** consist of data about the current project of concern.

A Matrix of $n \times m$ elements indicating the characteristics of each project in the HT.

$$\mathbf{A} = \begin{pmatrix} C_{11} & C_{12} & \cdots & C_{1m} \\ C_{21} & C_{21} & \cdots & C_{2m} \\ \cdots & \cdots & & \cdots \\ C_{n1} & C_{n2} & \cdots & C_{nm} \end{pmatrix}$$

C_{ij} = Characteristic j for the project i

Two possible values:

1: The project has the characteristic C_j

0: Otherwise

By using positive values greater than zero rather than just the value 1 in **A**, the

company's experts could assign relative weights to the *ACT* values on a project-independent basis. Values of 1 carry the unweighted *ACT* values forward from the HT into the estimates of the future *ACT*.

T Matrix of $n \times 1$ elements indicating the average annual change traffic in each project $\mathbf{T} = (ACT_1\ ACT_2\ \ldots\ ACT_n)^{\mathrm{T}}$. ACT_i = annual change traffic for project i where the superscript index 'T' expresses the operation *transposed matrix*.

C Matrix of $1 \times m$ elements indicating the characteristics of the current project of concern. The provision of these data requires the intervention of expert personnel. This is when the group of characteristics will be revised. Modification of this group requires updating the HT, revising the characteristics of all the projects in the HT.

$$\mathbf{C} = (c_1, c_2, \ldots, c_m)$$

c_j = Characteristic j for the current project

Two possible values:

1: The project has the characteristic

0: Otherwise

B Matrix of $n \times 1$ elements indicating the rate of coincidence the current project has in relation to each project of the HT—that is, the sum of characteristics they have in common (each characteristic contributes to the sum according to its historical *ACT* values).

$$\mathbf{B} = \mathbf{A} * \mathbf{C}^{\mathrm{T}} \tag{19}$$

where the symbol * indicates the product of matrixes.

Using these matrix definitions, the future *ACT* is then estimated by the following formula:

$$ACT = \frac{\mathbf{B} * \mathbf{T}^{\mathrm{T}}}{\mathbf{B}^{\mathrm{T}} * \mathbf{B}} \tag{20}$$

In this way each project is involved in calculating the estimate as far as its characteristics match those of the project under study.

3.5.4. Application of the model

Once we have the maintainability functions (F_{C}, F_{M} and F_{T}) and the estimated future *ACT*, we can estimate the maintenance cost in man-months for the project of concern by substituting values into expressions (13)–(16). This assumes we have the following data for the current project of concern:

MM_{DEV}: development cost expressed in man-months,
X_{U}: understandability metric,
X_{M}: modifiability metric, and
X_{T}: testability metric.

4. CASE STUDY

4.1. Introduction

In this case study, the model has been applied on three projects: a project to develop a software package for accounting management (P_1) and two for commercial management (P_2 and P_3). The case study has been simplified by taking into consideration only one of the maintainability components, understandability. Studying the modifiability and the testability would be identical in form.

Table 2 is an HT which shows the three projects studied. All the data have been measured except the understandability index which is obtained from applying expression (17). For simplicity, we present data for only two project characteristics, C_1 and C_2. We observed in the historical data for this company that the *ACT* is influenced by the application area (accounting or commercial), and that these two characteristics are of equal importance. Therefore:

C_1—accounting management, and
C_2—commercial management.

The project studied, P_4, is a project to develop a software package for commercial management. Its development is complete. The cost of development was 57 man-months, and the understandability metric value, X_U, is 17.

4.2. Understandability function

To facilitate determining the understandability function F_C, we first simplify the computational procedure. Using the method of least squares (see expression (8)), we have:

$$\text{minimize} \sum_{i=1}^{n} (I_{U_i} - F(X_{U_i})) \tag{21}$$

Because of its simplicity, in the first place we will consider the linear function adjustment case:

Table 2. History table for case study

Project	C_1	C_2	*ACT*	MM_{DEV}	X_U	MM_U	I_U
P_1	1	0	0·23	48	14	6·6	0·60
P_2	0	1	0·29	72	11	15·7	0·75
P_3	0	1	0·30	24	15	3·8	0·53

$$\text{minimize} \sum_{i=1}^{n} (I_{U_i} - (a + bX_{U_i})) \tag{22}$$

Partially differentiating this expression according to a and equalizing to zero, and also partially differentiating the expression according to b and equalizing to zero, we obtain the following equation system, in which we simplify the notation with summation limits from $i = 1$ to n.

$$\left.\begin{aligned} \sum I_{U_i} &= aN + b \sum X_{U_i} \qquad (23)\\ \sum X_{U_i} I_{U_i} &= a \sum X_{U_i} + b \sum X_{U_i}^2 \qquad (24) \end{aligned}\right\}$$

Now we consider the exponential function:

$$I_{U_i} = ae^{bX_{U_i}} \tag{25}$$

Applying logarithms, we obtain:

$$\text{Ln } I_{U_i} = \text{Ln } a + bX_{U_i} \tag{26}$$

So we now again have a linear function, where the independent variable is X_{Ci} and the dependent variable is $Ln\ I_{Ci}$. Making the change in the variable $I'_{Ui} = \text{Ln } I_{Ui}$ as well as $a' = \text{Ln } a$, the following system is obtained:

$$\left.\begin{aligned} \sum I'_{U_i} &= a'N + b \sum X_{U_i} \qquad (27)\\ \sum X_{U_i} I'_{U_i} &= a' \sum X_{U_i} + b \sum X_{U_i}^2 \qquad (28) \end{aligned}\right\}$$

The data needed to solve this equation system are shown in Table 3.
Substituting values and solving the resulting equation system, we obtain the values:

$$a = 1{\cdot}86 \tag{29}$$

Table 3. Computation table for case study

Project	X_U	I_U	$I'_U = \text{Ln } I_U$	X^2_U	$X_U I'_U$
P_1	14	0·60	−0·51	196	−7·14
P_2	11	0·75	−0·29	121	−3·19
P_3	15	0·53	−0·63	225	−9·45
Sum	40	1·88	−1·43	542	−19·78

$$b = -0{\cdot}08 \tag{30}$$

So the understandability function obtained, F_C, is the following:

$$I_U = F_U(X_U) = 1{\cdot}86\, e^{-0{\cdot}08\, X_u} \tag{31}$$

4.3. Estimated annual change traffic (*ACT*)

We now build the matrixes **A**, **T**, **C** and **B** according to the procedure explained in Section 3.5.3.

$$\mathbf{A} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \quad \mathbf{T} = \begin{pmatrix} 0{\cdot}23 \\ 0{\cdot}29 \\ 0{\cdot}30 \end{pmatrix} \quad \mathbf{C} = (0 \;\; 1) \quad \mathbf{B} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \tag{32, 33, 34 and 35}$$

Applying the *ACT* calculation formula from expression (20) we can estimate the future *ACT* for this project P_4.

$$ACT = \frac{\mathbf{B} * \mathbf{T}^T}{\mathbf{B}^T * \mathbf{B}} = \frac{0{\cdot}29 + 0{\cdot}30}{2} = 0{\cdot}30 \tag{36}$$

4.4. Estimated understanding cost (MM_U)

We can now calculate the estimated understanding cost expressed in man-months in the maintenance stage by using expression (9) which was:

$$MM_U = ACT \bullet MM_{DEV} \bullet I_U \tag{9}$$

Since the metrics shown previously for the P_4 project are:

$$MM_{DEV} = 57\ MM \tag{37}$$

$$X_U = 17 \tag{38}$$

and I_U is given by the function in expression (31) which was

$$I_U = F_U(X_U) = 1{\cdot}86\, e^{-0{\cdot}08\, X_u} \tag{31}$$

we have the following result:

$$MM_U = 0{\cdot}30 \bullet 57 \bullet 1{\cdot}86 \bullet e^{-0{\cdot}08 \bullet 17} = 8{\cdot}16 \text{ man-months.} \tag{39}$$

As previously stated, in this case study only one maintainability component has been

reported. For the other two components, the procedure would follow the same pattern. The estimated cost of maintenance expressed in man-months could be obtained from the sum of the three costs: understanding (MM_U), modifying (MM_M) and testing (MM_T).

5. CONCLUSIONS

Maintainability is a quality factor to be taken into consideration when estimating the cost of the maintenance stage in a software project. For this reason a factor for indicating the maintainability of a software product must be part of the calculation of this estimation. This factor we call the maintainability index. The interdependence between this index and a set of software metrics, which represent maintainability characteristics, is of great interest. The main element of this research is historical data from previous projects, an indispensable element for all activities including making estimations. In this paper the problems of estimating the cost of the maintenance process have been solved with our model, using data collected from previous projects. By applying the proposed model and procedures to these historical data, control over current and future maintainability can be improved, and thereby unnecessary and unproductive maintenance costs can be avoided.

References

Barranco-García, M. J. and Granja-Alvarez, J. C. (1996) 'Maintainability as a key factor in maintenance productivity: a case study', in *Proceedings of the International Conference on Software Maintenance*, Monterey, CA, IEEE Computer Society Press, Los Alamitos CA, pp. 87–93.

Boehm, B. W. (1976) 'Software engineering', *IEEE Transactions on Computers*, **C-25**(12), 1226–1241.

Boehm, B. W. (1979) 'Software engineering: R&D trends and defense needs', in Wegner, P. (Ed), *Research Directions in Software Technology*, MIT Press, Cambridge, MA, pp. 44–86.

Boehm, B. W. (1981) *Software Engineering Economics*, Prentice-Hall, Inc., Englewood Cliffs, NJ. 767 pp.

Boehm, B. W. (1987) 'Improving software productivity', *Computer*, **20**(9), 43–57.

Canning, R. G. (1972) 'That maintenance "iceberg"', *EDP Analyzer*, **10(**10), 1–14.

Chapin, N. (1987) 'The job of software maintenance', in *Proceedings of the Conference on Software Maintenance—1987*, Texas, IEEE Computer Society Press, Los Alamitos, CA, pp. 4–12.

Chapin, N. (1988) 'Software maintenance life cycle', in *Proceedings of the Conference on Software Maintenance—1988*, Arizona, IEEE Computer Society Press, Los Alamitos, CA, pp. 6–13.

Frost, D. (1990) 'Software maintenance and modifiability', in *Proceedings Software Maintenance and Computers*, San Diego, CA, IEEE Computer Society Press Tutorial, Los Alamitos, CA, pp. 187–192.

Harrison, W. and Cook, C. (1990) 'Insights on improving the maintenance process through software measurement', in *Proceedings of the Conference on Software Maintenance—1990*, San Diego, CA, IEEE Computer Society Press, Los Alamitos, CA, pp. 37–46.

Layzell, P. J. and Loucopoulos, P. (1988) 'A rule-based approach to the construction and evolution of business information systems', in *Proceedings of the Conference on Software Maintenance—1988*, Arizona, IEEE Computer Society Press, Los Alamitos, CA, pp. 258–264.

Lewis, J. and Henry, S. (1989) 'A methodology for integrating maintainability using software metrics', in *Proceedings of the Conference on Software Maintenance—1989*, Miami, FL, IEEE Computer Society Press, Los Alamitos, CA, pp. 32–39.

Musa, J. D., Iannino, A. and Okumoto, K. (1990) *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill, Inc., New York, NY. 291 pp.

Nosek, J. T. and Palvia, P. (1990) 'Software maintenance management: changes in the last decade', *Journal of Software Maintenance: Research and Practice*, **2**(3), 157–174.

Peercy, D. E. (1996) 'Book review of "*Improving the Maintainability of Software*"', *Journal of Software Maintenance: Research and Practice*, **8**(5), 345–356.
Schneidewind, N. F. (1987) 'The state of software maintenance', *Transactions on Software Engineering*, **SE-13(**3), 303–310.
Skramstad, T. and Khan, Md. K. (1992) 'A redefined life cycle model for improved maintenance', in *Proceedings of the Conference on Software Maintenance*, Orlando, FL, IEEE Computer Society Press, Los Alamitos, CA, pp. 193–197.
Swanson, E. B. (1976) 'The dimensions of maintenance', in *Proceedings of the 2nd International Conference on Software Engineering*, CL, IEEE Computer Society Press, Los Alamitos, CA, pp. 492–497.
Swanson, E. B. and Chapin, N. (1995) 'Interview with E. Burton Swanson', *Journal of Software Maintenance: Research and Practice*, **7**(5), 303–315.
Wallmüller, E. (1994) *Software Quality Assurance: A Practical Approach*, Prentice-Hall International (U.K.) Ltd., Hemel Hempstead, Herts, U.K., 279 pp.
Welker, K. D., Oman, P. W. and Atkinson, G. G. (1997) 'Development and application of an automated source code maintainability index', *Journal of Software Maintenance: Research and Practice*, **9**(3), 127–159.
West, R. (1993) *Improving the Maintainability of Software*, CCTA, HMSO, London. 104 pp.

Authors' biographies:

Juan Carlos Granja-Alvarez is Professor of Languages and Information Systems at the University of Granada in Spain where he heads the research group on Languages and Information Systems and Software Engineering. He is the author of three books and half a dozen papers. His Ph.D. in Computer Science is from the Politechnique University of Madrid. His E-mail address is: jcgranja@goliat.urg.es

Manuel José Barranco-García is a Group Leader in software engineering at a software company, Microjicsa, in Jaén, Spain. He has served as Professor of Management Information Systems at the National University of Distant Education in Spain, and has had substantial experience in software maintenance and development, and in project management. He holds a Ph.D in Computer Science from the Politechnique University in Madrid.

